The K9 On-Board Rover Architecture

John L. Bresina Maria Bualat Michael Fair* Richard Washington† Anne Wright*
Autonomy and Robotics Area, NASA Ames Research Center, Moffett Field, CA 94035 USA
{ jbresina | mbualat | mfair | rwashington | awright } @arc.nasa.gov

1 Introduction

This paper describes the software architecture of NASA Ames Research Center's K9 rover. The goal of the on-board software architecture team was to develop a modular, flexible framework that would allow both high- and low-level control of the K9 hardware. Examples of low-level control are the simple drive or pan/tilt commands which are handled by the resource managers, and examples of high-level control are the command sequences which are handled by the conditional executive. In between these two control levels are complex behavioral commands which are handled by the pilot, such as drive to goal with obstacle avoidance or visually servo to a target.

This paper presents the design of the architecture as of Fall 2000. We describe the state of the architecture implementation as well as its current evolution. An early version of the architecture was used for K9 operations during a dual-rover field experiment conducted by NASA Ames Research Center (ARC) and the Jet Propulsion Laboratory (JPL) from May 14 to May 16, 2000 [1].

2 The K9 Rover

The K9 Rover (see Figure 1) is a 6-wheel rocker-bogey chassis outfitted with electronics and instruments appropriate for supporting research relevant to remote science exploration and autonomous operations. The main CPU is a 166 MHz PC104+ mobile Pentium MMX single board computer running the Linux operating system. An auxiliary microprocessor communicates with the main CPU over a serial port and controls power switching as well as other I/O processing. The motion/navigation system consists of motor controllers for the wheels and pan/tilt unit, a compass, and an inertial measurement unit. These system components communicate with the main CPU over serial ports. The camera system consists of a stereo pair of high resolution SCSI filter-wheel cameras, and up to eight multiplexed stereo pairs of low resolution RS-170 cameras.

During field experiments, we have supported the testing of additional instruments on board the K9 rover. The Laser Induced Breakdown Spectrometer from Los Alamos National Laboratory and the flight panoramic camera from Ball Aerospace were tested during the May 2000 field experiment. Work is on-going to support three additional instruments: the CHAMP microscopic imager from the University of Colorado, Boulder, an arm-mounted, Raman spectrometer from Detection Limit, Inc./ Montana State University-Bozeman, and a pseudolite GPS system from Stanford University.

3 Architecture

The K9 on-board software architecture consists of 5 types of modules: device drivers, resource managers, behaviors/data processors, pilot, and exec. Figure 2 illustrates the architectural elements and their interactions, and the following subsections describe the individual components.

*NASA contractor with QSS.
†NASA contractor with RIACS.

Figure 1: The K9 Rover.

3.1 Device Drivers

At the lowest layer of the structure are the device drivers. These modules directly control the hardware devices of the robot, including framegrabbers, motors, and buses. Each device driver is controlled by a resource manager and cannot be commanded directly by the rover operator or plan executive.

3.2 Resource Managers

In the next layer up from the device drivers are the resource managers. The current set of resource managers include the base manager, the pan/tilt manager, and the vision manager. The base manager drives the rover's wheels, and the vision manager controls image acquisition. In the future this set will also include an arm manager and instrument managers, such as spectrometer or microscopic camera managers. Resource managers can be directly commanded using command messages or direct method calls. Command messages are transmitted by NDDS, a commercial publish/subscribe communication framework for real-time systems, based on an underlying datagram model.

3.3 Rover Behaviors and Data Processors

The rover behaviors and data processors comprise another type of module in the K9 software architecture. This set of modules currently includes obstacle avoidance and path planning, and in the near-term will also include visual tracking and stereo processing. Behaviors cannot be commanded directly by the rover operator nor the executive module; they are instead indirectly controlled via the pilot. This is a relatively generic module type of the architecture, in that the behavior modules require little knowledge about the details of the robotic platform performing the sensing and actuation for the module. As an example, consider the obstacle avoidance module. The inputs to this module are a range map of the current environment, the desired goal, and knowledge of the maximum allowable obstacle size and minimum obstacle spacing. Based on these inputs, the obtacle-avoidance module will output a preferred, safe driving direction.

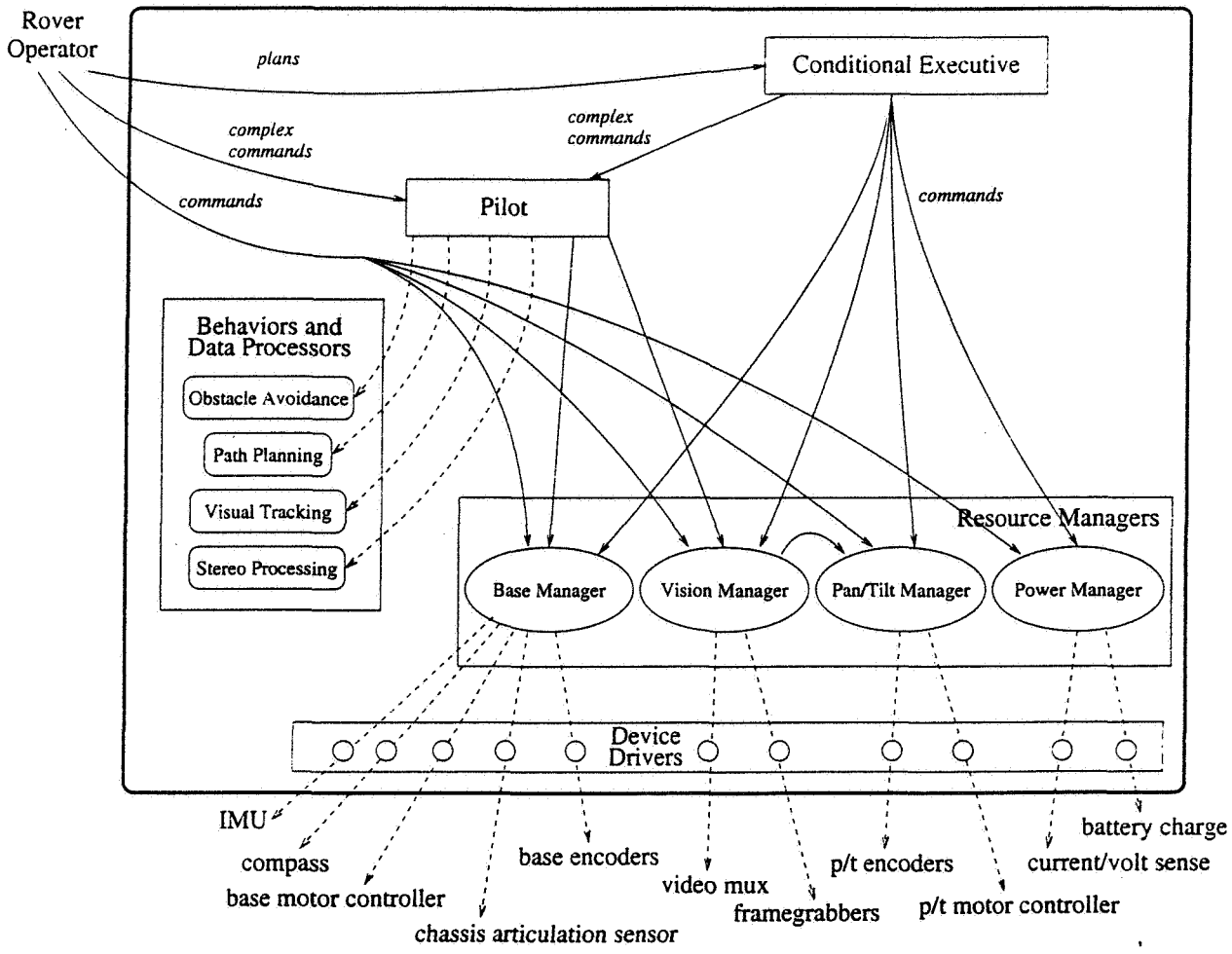

Figure 2: The K9 Rover Architecture. The rover can be commanded at the level of plans, complex commands, or low-level commands. The plans in turn may consist of complex and low-level commands, with additional temporal, resource, and state constraints. Solid lines indicate public interfaces, which are accessible to rover operators and executive control; dashed lines indicate private interfaces, which are accessible only to the controlling module.

Action:	(drive 10.0 0.05)
Wait-for conditions:	(time 10 300) (time +5 +20) (resource energy 5)
Start conditions:	(time 10 300) (time +5 +20) (resource energy 5) (resource storage 100) (rover-state :mechanical-state :ok) (rover-state :distance-traveled ?start-distance)
Maintain conditions:	(resource energy 2)
End conditions:	(rover-state :distance-traveled ?end-distance) (> ?end-distance ?start-distance)
Expectations:	(duration 100 10) (energy 2)
Continue-on-failure:	False

Figure 3: A CRL action. An action may have conditions that must hold before, during, and after execution. Conditions may be based on temporal, resource, or state information.

3.4 Pilot

The pilot is the next higher level of abstraction within the K9 software architecture. The pilot is the hub of the hardware control software, as it is the link between the behaviors and the resource managers. Complex commands, such as "visually servo to a target while avoiding obstacles", are directed to the pilot either by direct method calls from the executive or *via* a single NDDS command. The pilot then decomposes the complex command into a set of more specific commands to the resource managers and requests for information from the behaviors and data processors. In the example of visually servoing to a target while avoiding obstacles, the pilot commands the vision manager to acquire a stereo pair of images from the front hazard-avoidance cameras ("hazcams") and a stereo pair of images from the mast-mounted navigation cameras ("navcams"). The pilot then passes the hazcam image pair to the stereo processing component to obtain a range map. The pilot also passes the navcam image pair, along with a template image of the target, to the visual tracking module. The tracking module outputs a target location that is input to the obstacle avoidance module, along with the range map. The obstacle avoidance module outputs the safe drive direction, which is then used by the pilot to command the base manager to move K9.

3.5 Conditional Executive

At the highest layer of the structure is the conditional executive. The rover operator creates a command plan that contains commands for specific K9 subsystems; for example, mobility and cameras. The commands are written to an uplink file in the Contingent Rover Language (CRL).

CRL is a flexible, conditional sequence language that allows for execution uncertainty. CRL expresses time constraints and state constraints on the plan, but allows flexibility in the precise time that actions must be executed. Constraints include conditions that must hold before, during, and after actions are executed: failure of these conditions leads to an execution failure and potential plan adaptation. Figure 3 illustrates the types of conditions supported by CRL.

A primary feature of CRL is its support for contingent branches to handle potential problem points or opportunities in execution. The contingent branches and the flexible plan conditions allow a single plan to encode a large family of possible behaviors, thus providing responses to a wide range of situations.

The structure of the CRL plan language and its interpretation are completely domain-independent. Domain-dependent information is added by specifying a command dictionary, with command names and argument types,

and a command interface, which passes commands to the rover and return values and state information from the rover.

The conditional executive (CX) is responsible for interpreting the CRL command plan coming from ground control, checking run-time resource requirements and availability, monitoring plan execution, and potentially selecting alternative plan branches if the situation changes. At each branch point in the plan, CX may have multiple eligible options. CX chooses the option with the highest expected utility, computed over the remainder of the plan. From this utility and a model of the probabilities of various events (such as a traverse taking longer than anticipated), the expected utility of an entire branching plan can be estimated.

The initial estimate of the utility of executing actions is computed on the ground with respect to the expected resource and time availability. These estimates are only approximate — the actual time and resource availability is only known at execution time. To better handle uncertainties, we have developed methods to update plan utilities on board at runtime to reflect the current best estimate of action utilities [2]. In addition, the overall utility of the uplinked plan may change during execution, due to unforeseen science opportunities or changes in resource availability and demand profiles. In this case, the rover's behavior will be suboptimal unless its plan is revised. Furthermore, the scope of the plan may be exceeded due to failures. In this case, without plan revision, the rover must wait until a new plan is uplinked. We address the issues of plan suboptimality and plan failures by performing limited plan revision on board [3]. Our long-term strategy is to continue to increase rover on-board planning capabilities as flight processors become more powerful and as rover autonomy software becomes more acceptable.

4 Current Status and Future Directions

The K9 rover has been demonstrated in May 2000, as part of the joint JPL-Ames field test; it has also been demonstrated at our local test site at NASA Ames. We have a full suite of device drivers and resource managers for the onboard fans and power system components as well as the following onboard instruments and sensors: cameras, a pan-tilt head, a laser rangefinder, an inertial measurement unit, a compass, temperature sensors. We have recently installed an arm with 5 degrees of freedom, which will be used to deploy instruments on targets in order to perform *in-situ* science. The pilot has been demonstrated for use with obstacle avoidance and motor control. Work on integrating visual navigation and stereo processing is ongoing. The conditional executive has also been demonstrated for control of behaviors; we are currently extending the executive to control the pilot's new capabilities. We are also integrating on-board science analysis algorithms [4], which will be used by the executive to increase the rover's scientific productivity. One future direction of the K9 software architecture is to explore its compatibility with the CLARAty software architecture from JPL [5]. As our first step in this effort, the arm control is being developed within the CLARAty framework.

Acknowledgments: In addition to the authors, the K9 rover team included K. Bass, J. Bowman, L. Edwards, L. Flueckiger, L. Kobayashi, L. Nguyen, C. Stoker, and H. Thomas.

References

[1] J. L. Bresina, M. G Bualat, L. J. Edwards, R. M. Washington, and A. R. Wright. K9 operations in May '00 dual-rover field experiment. In *Proceedings of the Sixth International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS-2001)*, 2001.

[2] J. L. Bresina and R. Washington. Expected utility distributions for flexible, contingent execution. In *Proceedings of the AAAI-2000 Workshop: Representation Issues for Real-World Planning Systems*, 2000.

[3] J. L. Bresina and R. Washington. Robustness via run-time adaptation of contingent plans. In *Proceedings of the AAAI-2001 Spring Syposium: Robust Autonomy*, 2001.

[4] V. C. Gulick, R. L. Morris, M. Ruzon, and T. L. Roush. Autonomous science analysis of digital images for mars sample return and beyond. In *The 30th Lunar Planetary Science Conference*, 1999.

[5] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The CLARAty architecture for robotic autonomy. In *Proceedings of the 2001 IEEE Aerospace Conference*, March 2001.